ARPA ORDER NO. 189-1

MEMORANDUM
RM-6001-ARPA
SEPTEMBER 1969

THE GRAIL LANGUAGE AND OPERATIONS

T .O. Ellis, J. F. Heafner and W. L. Sibley

PREPARED FOR:
ADVANCED RESEARCH PROJECTS AGENCY

The RAND Corporation
SANTA MONICA • CALIFORNIA

This study is presented as a competent treatment of the subject, worthy of publication. The Rand Corporation vouches for the quality of the research, without necessarily endorsing the opinions and conclusions of the authors.

Published by The RAND Corporation

ARPA ORDER NO. 189-1

MEMORANDUM
RM-6001-ARPA
SEPTEMBER 1969

THE GRAIL LANGUAGE AND OPERATIONS

T. O. Ellis, J. F. Heafner and W. L. Sibley

This research is supported by the Advanced Research Projects Agency under Contract No. DAHC15 67 C 0111. Views or conclusions contained in this study should not be interpreted as representing the official opinion or policy of ARPA.

The RAND Corporation
1700 MAIN ST • SANTA MONICA • CALIFORNIA • 90406

PREFACE

This Memorandum is the second of a three-part[†] final report on the GRAIL (GRAphical Input Language) Project sponsored by the Advanced Research Projects Agency of the Department of Defense. The study is an integral part of both RAND's and the client's overall program to explore man-machine communications.

The GRAIL computer flowchart language and its use are highlighted in this Memorandum. A man, using a RAND Tablet/Stylus and a CRT display, may specify and edit a computer program via flowcharts and then execute it. The system provides relevant feedback on the CRT display.

[†]See also by the same authors: *The GRAIL Project: An Experiment in Man-Machine Communications*, The RAND Corporation, RM-5999-ARPA, September 1969; *The GRAIL System Implementation*, The RAND Corporation, RM-6002-ARPA, September 1969.

SUMMARY

This Memorandum highlights the GRAIL (GRAphical Input Language) flowchart language and its use. A man using a RAND Tablet/Stylus and a random deflection CRT display may draw flowchart symbols, edit and rearrange them on the display surface, and connect them appropriately to form a meaningful program. He may also execute the program while controlling its execution rate and the amount and content of information presented to him. The system interprets, in real-time, the man's hand-drawn figures, characters, and other stylus gestures to provide germane responses on the display surface.

Operations were governed by the principles that the system should be responsive, easy to understand, easy to use, and powerful. The allowable symbol set is given along with the rules for editing text. Other control functions embraced by the system are also described; e.g., virtual button-pushing and picture-editing manipulations.

The language organization centers on sequential control flow and nested subroutines coupled with flowcharts to relate their interdependence pictorially. These notions help the man to structure his program and to envision graphically its organization in two dimensions. The Memorandum describes the symbolism, its structure, and its labeling conventions.

The unpredictable dynamic actions taken by the man led to symbolizing parallel processing and task synchronization to provide good response times; this extended symbolism is also described.

Since the man's intentions are determined from stylus movement alone, a method was established to deduce that intent by associating the stylus movement with the elements of a displayed picture and their internal representations. The technique, as it applies to the geometry of flowchart symbols, is discussed as well as the merits of different kinds of responses.

The flowchart language and the supporting operating system were designed for the exploration of interactive graphical communication techniques.

ACKNOWLEDGMENTS

The authors would like to thank the many reviewers of this Memorandum for their suggestions as well as the following people for their discussion of this work: G. F. Groner, R. Patrick, J. C. Shaw, and R. Turn.

CONTENTS

PREFACE iii

SUMMARY v

ACKNOWLEDGMENTS vii

FIGURES xi

Section
I. INTRODUCTION 1

II. OPERATIONS 2
Operating Principles 2
The Stylus, Inking Feedback, and Other Responses 3
Symbol Recognition 3
Text Editing 3
Control Functions 5
Execution 7

III. FLOWCHART LANGUAGE 9
Processes, Planes, and Flow of Control .. 9
Frames and Implied Control Flow 11
Closed Planes and Files 11
Basic Symbolism 11
Parallel Processing and Extended Symbolism 14
Code Planes 15
Label Conventions 15
Summary of Organization 16

IV. DISCUSSION 20

REFERENCES 23

FIGURES

1. Symbol Ink Tracks 4
2. Conceptual Flowchart Organization 10
3. Flowchart Symbols 12
4. Parameters 17
5. Translation 18

I. INTRODUCTION

The GRAIL (GRAphical Input Language) research experiment was designed to facilitate problemsolving by providing a useful interface between man and machine. Specifically, the project investigated techniques for the real-time interpretation of free-hand gestures (on a RAND Tablet), display representation methods, and their application to a significant problem area--constructing computer programs via flowcharting.

The system permits construction, editing, interpretive execution, compilation, debugging, documentation, and execution of computer programs specified by flowcharts. The communication language is structured to assist the man in problem formulation by allowing specification of a problem, editing of its constructs, and validating its representation. Accurate and intelligible documentation directly results from the problem statement in GRAIL; and means exist to exercise the problem description to find solutions.

This Memorandum, the second of three (see Refs. 1 and 2), address the language and operations that were developed to exemplify the above features as problemsolving aids.

II. OPERATIONS

OPERATING PRINCIPLES

Several aspects of the operating environment contributed to the GRAIL development. The hardware[†] was available largely on an exclusive basis, permitting GRAIL to function as an operating system. The RAND Tablet/CRT display offered inherent two-dimensionality. These aspects helped realize the goals for user operation--viz., responsiveness, ease of understanding, ease of use, and power.

To make the operating procedures apparent, one must insure consistent representation, manipulation, and response. The pictures are uncluttered and their formats are similar. Symbols the man constructs in a pattern familiar to him are changed by the system only at a time when his attention is focused on them. Consistent placement of virtual buttons[‡] and fully descriptive labels rather than abbreviations, make manipulations obvious. Responses typically have unique meanings, but may occasionally have generic meanings. Similar responses may occur under different conditions, but in any situation a response has a specific meaning that is apparent with a minimum of user sophistication.

Use is made easier through the on-line and two-dimensional aspects. Two-dimensionality allows directness; i.e., operations are in-place and are effected with a single input instrument, the tablet stylus. On-lineness allows interaction; e.g., the system can check validity by interpreting and responding to the man's gestures in real time.

[†]The hardware was an IBM System/360 Model 40, two IBM 2311 disk drives, a Burroughs Corp. prototype CRT, and a RAND Tablet with match circuitry and direct feedback.

[‡]Virtual buttons are outlined as labeled areas on the display surface whose corresponding tablet position may be pressed by the stylus to invoke some function.

The operational techniques described below were developed with these operating principles in mind.

THE STYLUS, INKING FEEDBACK, AND OTHER RESPONSES

Response to stylus input depends upon context--i.e., the particular picture displayed and the local area addressed on that picture--and, to some extent, on the time and direction of the stylus motion. For example, ink is supplied when the man is drawing flowchart symbols and printing characters, but not when he invokes a virtual button or moves existing figures. Other responses are English-language statements, changes in display intensity, and picture modification.

SYMBOL RECOGNITION

One fundamental facility of the man-computer interface is automatic recognition of appropriate symbols. The GRAIL system allows the man to print text and draw flowchart symbols naturally; the system recognizes them accurately in real-time. The recognizable symbol set includes the upper-case English alphabet, the numerals, seventeen special symbols, a scrubbing motion (≋) used as an erasure, and six flowchart symbols--circle, rectangle, triangle, trapezoid, ellipse, and lozenge. Symbols are drawn with a variety of stylus motions. The system applies tests for time, space, and features to determine completion of a symbol.

The tablet data corresponding to the displayed ink track must be accurately interpreted as one of the symbols. Figure 1 shows a set of ink tracks corresponding to stylus input recognized as the symbol 3.

TEXT EDITING

Input and modification of text are fundamental uses of the graphic console. This on-line interface actually has

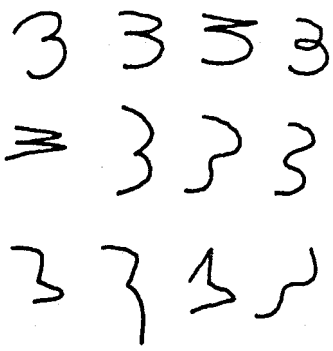


Fig. 1--Symbol Ink Tracks

fewer constraints than the familiar blackboard or pencil and paper because of the dynamics and memory of the system; e.g., one may replace one character with another by over-printing, which requires fewer motions than with pencil and paper because erasure is implied.

GRAIL's text-editing features are: character placement and replacement, character-string insertions, line insertions, character and character-string deletions, and line deletions. One may edit in-place with few constraints on the sequence of motions. No positional maneuvers (e.g., moving a cursor) are required.

An alphanumeric or special symbol may be handprinted in-place (*character placement*); when completed, its ink track is replaced by a hardware-generated character. When a character is printed over an existing character (*character replacement*), the system replaces the previous character with the newly-recognized character.

One erases by scrubbing (as in erasing a blackboard) over the character(s) to be deleted. Any number of characters within a line may be erased by a single scrubbing

motion; the horizontal extent of the scrub defines the boundary of the erasure. Erasure of blanks shifts the remaining characters (to the right of the blanks) leftward over the erased blanks.

One may insert a string of characters between two characters by drawing a caret (^) between them. The display dims and a dashed line appears above and to the right of the caret to indicate the maximum character-string length that may be inserted. A new string printed above the dashed lines by the man appears at normal intensity. When the caret is erased, the dashed line vanishes along with it, and the display returns to normal intensity with the new string inserted.

One may insert blank lines between existing lines by drawing a '>' symbol in the left margin. The lines below are moved down by one line space and a new blank line is inserted. The previously displayed bottom line disappears.

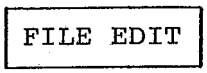
Erasing all the characters on the line and then erasing again on the blank line deletes the line entirely. The lines below then move up one line space.

Syntax analysis is performed on character strings where it is appropriate, and errors are indicated by brightening the entire line. The analysis is performed when the line is completed; i.e., when the stylus addresses some other area of the display, such as the next line.

CONTROL FUNCTIONS

Stylus-activated control functions are invoked through sensitive areas--some visual and some implied by their geometric relationship to displayed figures.

Such virtual buttons as the one shown below activate functions when the stylus is pressed in the button area



and then released. This particular operation replaces the current environment (and display) with a file-editing environment.

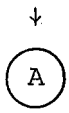
Other buttons activate operations that continue as long as the stylus switch is closed in the vicinity of the button. Consider the CRT surface a *viewing window* into a long scroll of text lines. The text may be moved page-by-page in either direction past the window as long as the stylus is down over the desired directional arrow (see below).

↑PAGE↓

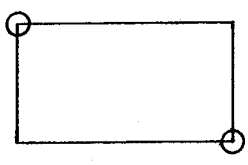
A *page* is the amount that may be displayed before the window at one time. The paging rate was empirically determined and depends on the kind of information being displayed.

A scrolling button, operated like the page button, moves the text vertically past the viewing window line-by-line rather than page-by-page. Again, motion ceases when the stylus is lifted or moved away from the directional arrows.

Control functions extend to the graphic domain as well; e.g., one may request a picture related to the circle shown below by pressing on it. (The frame to which the connector 'A' leads is displayed.)



One may move the rectangle shown below



by placing the stylus on the circle *handle* in the upper left and moving the stylus across the tablet surface. The rectangle will be dimmed initially and its displayed location will be updated continuously to match the stylus position. (In certain instances, character strings may be transported in a similar fashion.) One may alter the size and proportion of the rectangle via the handle in its lower right-hand corner. (All flowchart symbols may be moved, but only the rectangle may be changed in size.) Any text contents are treated as part of the figure.

The source for drawing a flow-line connection between two graphics is the perimeter of the source graphic, which immediately brightens when the stylus is placed on its perimeter. The flow line (composed of rather long line segments instead of ink vectors) remains (or disappears if appropriate) after the stylus is lifted from the destination graphic.

Stylus *hits* are detected on flow lines that bound no area as well as on symbols that do bound areas. The system employs the tablet/display match hardware (an analog comparison under program control between the stylus coordinates and the CRT beam deflection), similar to the light pen strike, to detect the hit. Flow lines brighten when addressed and may be erased via the scrub motion.

EXECUTION

The man may execute part or all of his program from the console either by compiling the processes and executing them at CPU speeds or by interpretive execution.

Execution of compiled processes is used typically after the processes have been constructed and debugged. The man specifies parameters, then views the results after execution is completed.

Interpretive execution, designed to be much more interactive, is used for debugging. The man controls execution

by starting, stopping, continuing, and terminating with simple, direct stylus actions. He controls execution rate in either single-step or variable mode (up to a display frame-swapping rate of about 30 ms/frame) as well as the amount and content of information presented on the display. Brightening the next graphic to be executed and scrolling the next code statement to the top of the viewing window shows the control flow through flowchart symbols and code statements, respectively. The man may overlay or delete the changing data-value display (parameter frame) at any time; therefore, he may view any change (data value or control step) to his program.

The information displayed during interpretive execution is exactly the same picture that the man constructed. In fact, the man frequently uses the overlay (e.g., parameters and flowchart) and split-screen (parameter and code statements) images during construction. For overlays, one picture is displayed at normal intensity and may be stylus addressed; the second picture, contrasting in format, is dim and may not be affected. For similarly formatted pictures, the screen is partitioned horizontally (split-screen) to present part of each--both at normal intensity and both addressable.

During interpretive execution, the flexible controls of information displayed, and the dynamic visual feedback aid debugging with the necessary information at the appropriate time.

III. FLOWCHART LANGUAGE

Important organizational concepts in the GRAIL system are the sequential flow of control, the hierarchy of subroutines, and the language (flow diagrams) for pictorially relating the organization within the concepts of the first two. The sequential nature of control allows the man to envision isolated processes that are adapted to specific functions--which, in turn, allow the organizer to think of the total program in terms of manageable subparts. The subroutine hierarchy emphasizes the notion of isolated processes even more strongly. Flow diagrams help the man to *picture* his control options and the relationship between processes by expressing these interrelationships in two dimensions.

PROCESSES, PLANES, AND FLOW OF CONTROL

Each computer *process* requires a definition. The main ideas and their interrelationships constitute a conceptual *plane*. The next level of detail for a particular notion constitutes another conceptual plane and so on, until the lowest level of detail has been explicitly expressed by appropriate computer-language statements or flowchart symbols. This collection of planes constitutes the process definition (see Fig. 2).

The process box, ROOT, in Fig. 2 represents an *instance* of use of a process. The process's progressive definition through several planes is shown. Note that no relationship exists between the plane defining ROOT and that defining SCALE, even though they both lie conceptually one level below the plane POLY. The instance provides the link from the plane on which the box appears, into the plane that defines the process. Control flows into the plane through a single *entry* and returns from the defining plane via an *exit*.

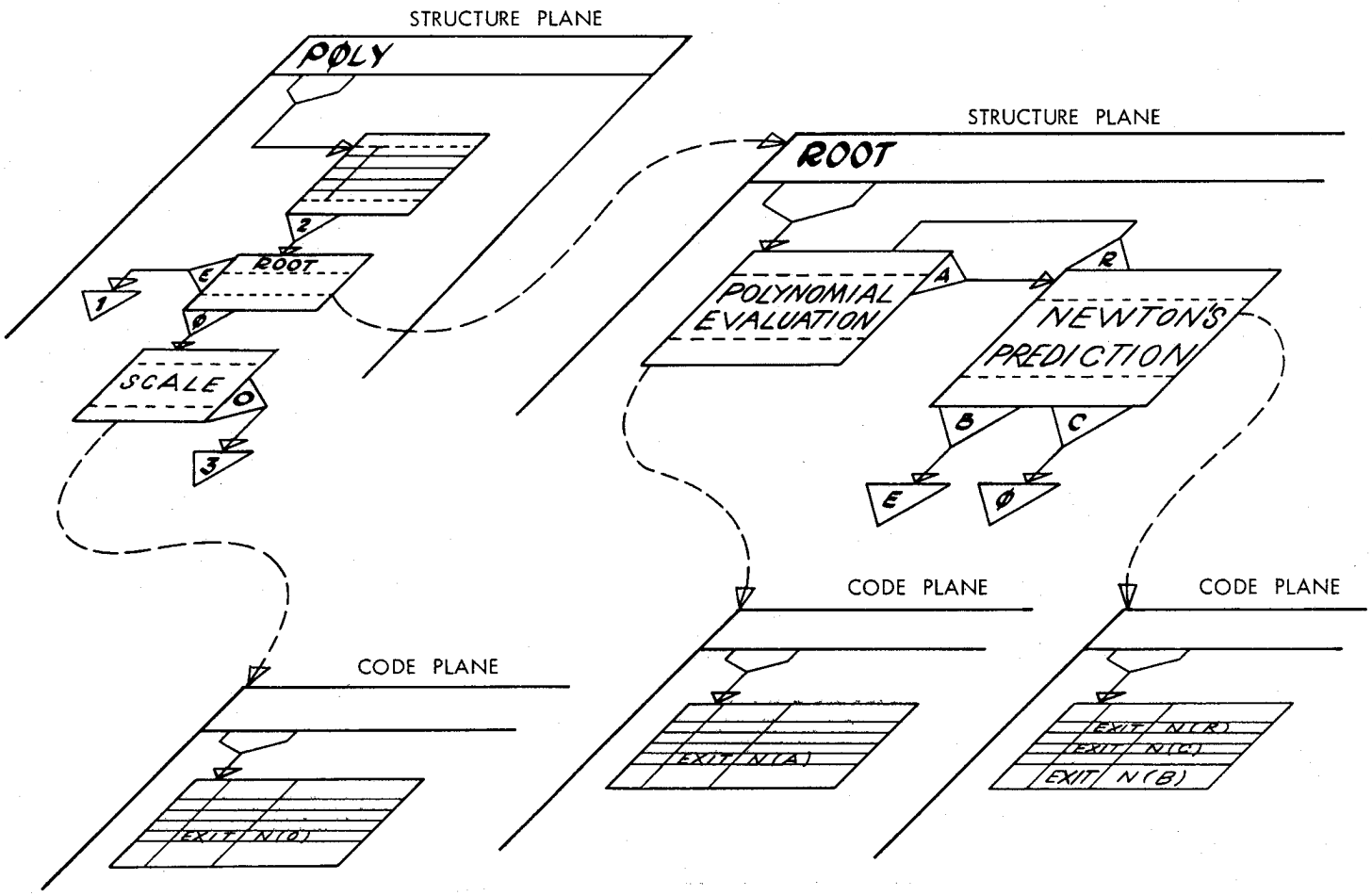


Fig. 2--Conceptual Flowchart Organization

On the plane, control flows from one process instance to the next via *flow lines*. After returning from the process definition, control flows from one instance to the next via a *decision* attached to the first process instance. The decision corresponds to a specific exit in the underlying definition. Exits and their corresponding decisions are associated by common labels.

FRAMES AND IMPLIED CONTROL FLOW

A plane is unlimited in two dimensions, but because of the display hardware limitations it is divided into a collection of display *frames* (pictures). The frames, logically but not geometrically connected, are the units for displaying and storing the pictorial representation of the flow diagram.

Within a frame, control flows from one flowchart element to another via flow lines. From frame to frame, control flows from a *connector instance* (see Fig. 3) on one frame to the corresponding *connector definition* on another frame in the plane. (Instance and definition are associated by label.)

CLOSED PLANES AND FILES

It is convenient to use multiple instances of a process in other processes, usually supplying different parameters for each instance. Each *parent process* containing an instance of the reentrant process definition (*daughter process*) provides, at execution time, a unique *context* (or operating environment) for each instance of the daughter process. A reentrant process definition must be *closed* by uniquely naming it. The collection of closed (labeled) *process definitions* form the man's *file* or program.

BASIC SYMBOLISM

Figure 3 illustrates the various flowchart symbols. Each plane has a single starting point (entry), characterized

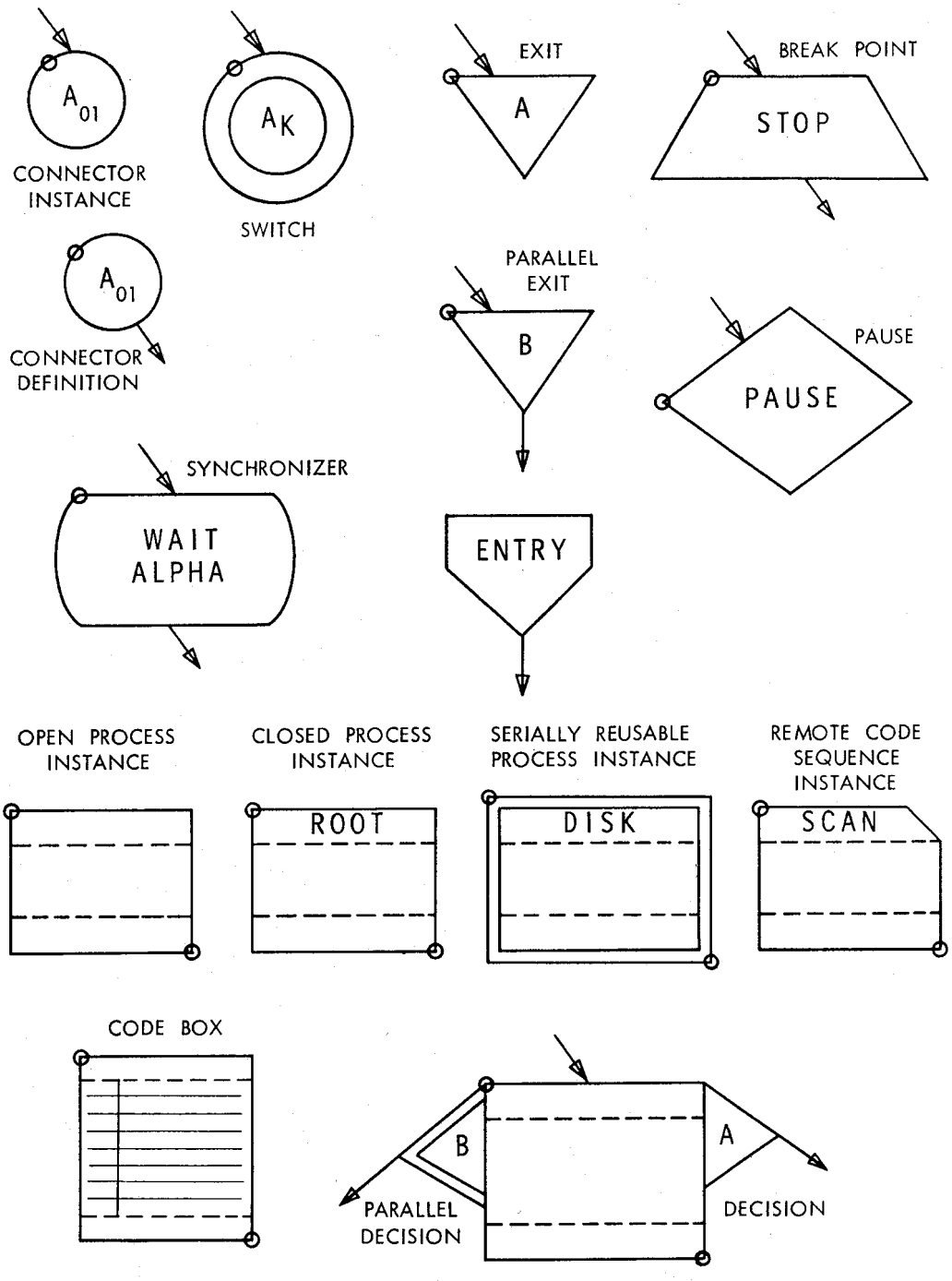


Fig. 3--Flowchart Symbols

by a single outgoing flow line and no incoming flow lines. The *terminal exit* is the return to the invoking instance from a plane definition; it has one or more incoming flow lines and no emanating flow. Control flows from the exit in the defining plane through the decision of the instance to the next graphic in the plane containing the instance. The decision has a single issuing flow line. The exit and the decision are paired by a common character label. Many exits may return to a single decision having that same unique label.

Five kinds of processes are tailored to particular uses:

1) The *open process* has a single instance and a single definition (unlabeled), similar to an in-line macro. The defining plane may be expressed as a flow diagram or detailed code.
2) The *code box* (unlabeled) has no underlying plane. The detailed statement definition (such as assembler language) appears within the box instance.
3) The *closed process* (labeled) consists of a flowchart or a code definition and may have any number of instances in any other planes, similar to the classical subroutine. It is reentrant and may be used recursively.
4) The *serially reusable process* (labeled) has one definition and any number of instances. It communicates with a single facility--e.g., a particular disk drive.
5) The *remote code sequence* (labeled) has a code definition and many instances. It is defined in terms of a code plane. It also transmits parameters differently from the closed process. It affords quick access to a simple function.

Connector instances and connector definitions provide inter- as well as intra-frame logical coupling. Any number of instances with flow lines incoming may exist, but only

one definition with a single outgoing flow line exists for a given connector label within a plane. Instances and their definition are associated through labels that are unique within a plane. The *switch* is another flowchart symbol (whose label reference is bound by the conceptual plane) for controlling flow among frames. It is a many-way branch with incoming flow lines. The result of executing a switch is to branch to the connector definition identified by a common label and by the current numerical value of the *index* associated with the switch.

The *break point* may have many incoming flow lines and one outgoing flow line. When it is encountered during interpretation, the system suspends control flow to allow the man to examine program and data values.

PARALLEL PROCESSING AND EXTENDED SYMBOLISM

The man may symbolize multiprogramming for tasks that are independent over some interval--a *parallel exit* initiates two tasks at some point in the control path. One task may be suspended (possibly for an I/O operation, resuming after its completion) without affecting the second. The parallel-processing symbolism is used for other functions as well as the external transmission of data.

Figure 3 (p. 12) illustrates the flowchart symbols for the parallel exit and *parallel decision*. The control flow from the decision is a higher priority task than the control flow from the exit in the plane containing the exit. With a single CPU, the lower priority task will not be initiated until the higher priority task is suspended or terminated. Parallel decisions and exits have the same one-decision-to-many-exits property as do the nonparallel decision and terminal exits. The parallel exit has one or more incoming flow lines and a single issuing flow.

A task terminates when control flow reaches a *pause*; then a suspended task is automatically initiated. The pause has incoming flow lines but no outgoing flow.

Synchronizing the control flow of parallel tasks is often necessary. The *synchronizer* has any number of incoming flow lines and a single emitting flow; labels within it identify an operation type and the name of a *program status group* (PSG). The PSG refers to information governing control flow, the task environment, and interrupt-inhibit status. Examples of operation types are WAIT and SET. WAIT suspends the task until another control flow encounters a SET identifying the same PSG. If the SET occurs before the WAIT, the task reaching the WAIT is not suspended. Note that when the SET occurs after the WAIT, the flow issuing from the SET has the higher priority--the opposite of the parallel exit/parallel decision priority.

CODE PLANES

Process definitions may be expressed as detailed code statements as well as flowchart symbols. Code planes may contain statements equivalent to the graphics--entry, exit, parallel exit, and break point--and an assembler language. A code plane--either closed or open--has no depth of program specification; i.e., it cannot contain any daughter-process instances.

LABEL CONVENTIONS

Labeling data, processes, connectors, or specific code statements is sometimes necessary (or often simply convenient). The man specifies the label type. Processes may use formal (dummy) labels--a specific label (data reference only) will replace the formal label for each invoked instance of the process. No global (external) data labels exist.

Connector labels (instances, definitions, and switches) are local (i.e., cannot be externally referenced) to the plane in which they appear. Similarly, decision/exit labels are local to the plane in which the exit label appears. Each

label appearing in a code statement is local to the plane in which it occurs.

Process names are not coincident with other labels. Each closed process within a file has a unique name--the only "global" names within a file.

Associated with each closed-process definition is a *parameter frame* containing both local and formal data labels (see Fig. 4). Initial values (for either constants or temporary data space) are specified for local data labels. The data space to which temporary labels refer exists only during execution of the process and is unique for each instance invoked.

The formal data labels represent input/output arguments transmitted to the definition by each instance of the process. Figure 5 shows the *translation frame* associated with each instance of a closed or serially reusable process. The formal label is made to correspond on the translation frame with the label of its real equivalent; i.e., its object-time data space (arguments are passed by name, not by value). They can be transmitted through as many processes as desired, but are ultimately translated into a local constant or a local temporary. Parameter transmission to a remote code sequence is specified on a translation frame, but symbolically named hardware registers replace formal labels.

SUMMARY OF ORGANIZATION

A man may have many files or programs. Each is a diagramatically ordered collection of closed-process definitions whose instances may appear in other processes. Each closed process is a collection of planes (open processes have only one instance for each definition). Each plane is a collection of frames implicitly coupled via connectors and may contain instances of other processes (open or closed). Each frame contains a collection of flowchart symbols or code statements. Associated with each closed process is a

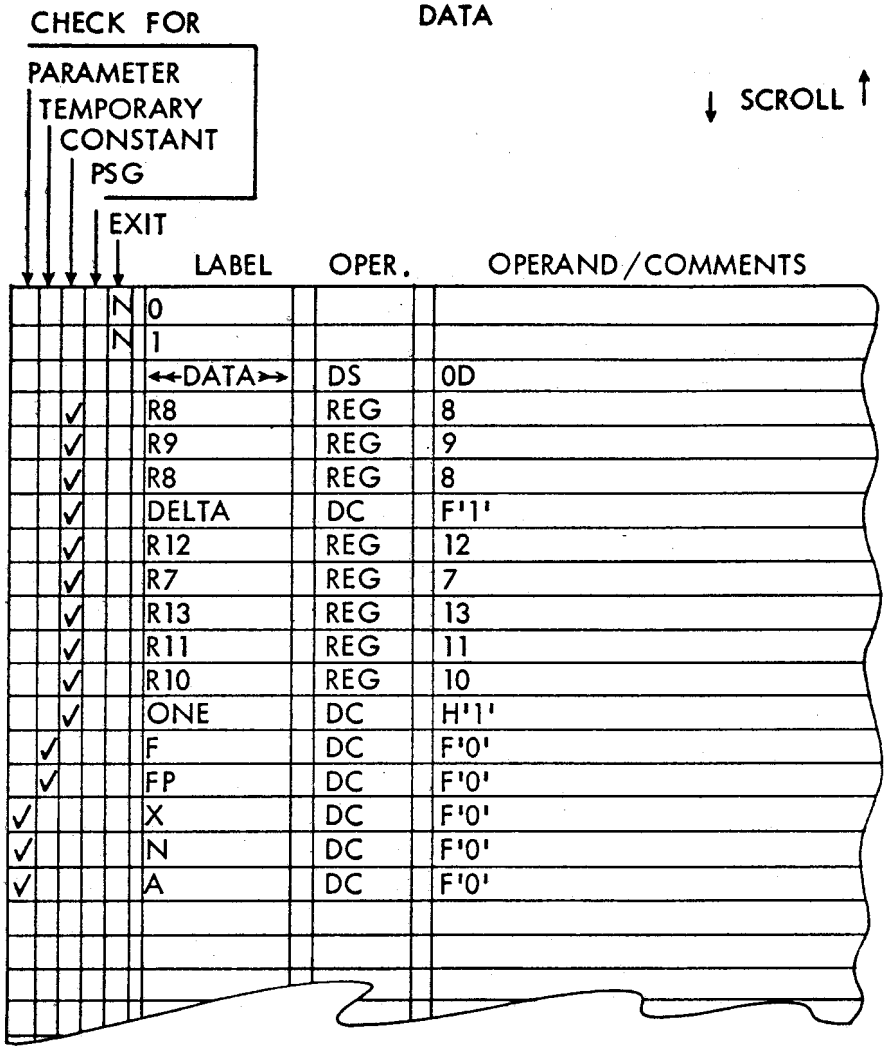


Fig. 4--Parameters

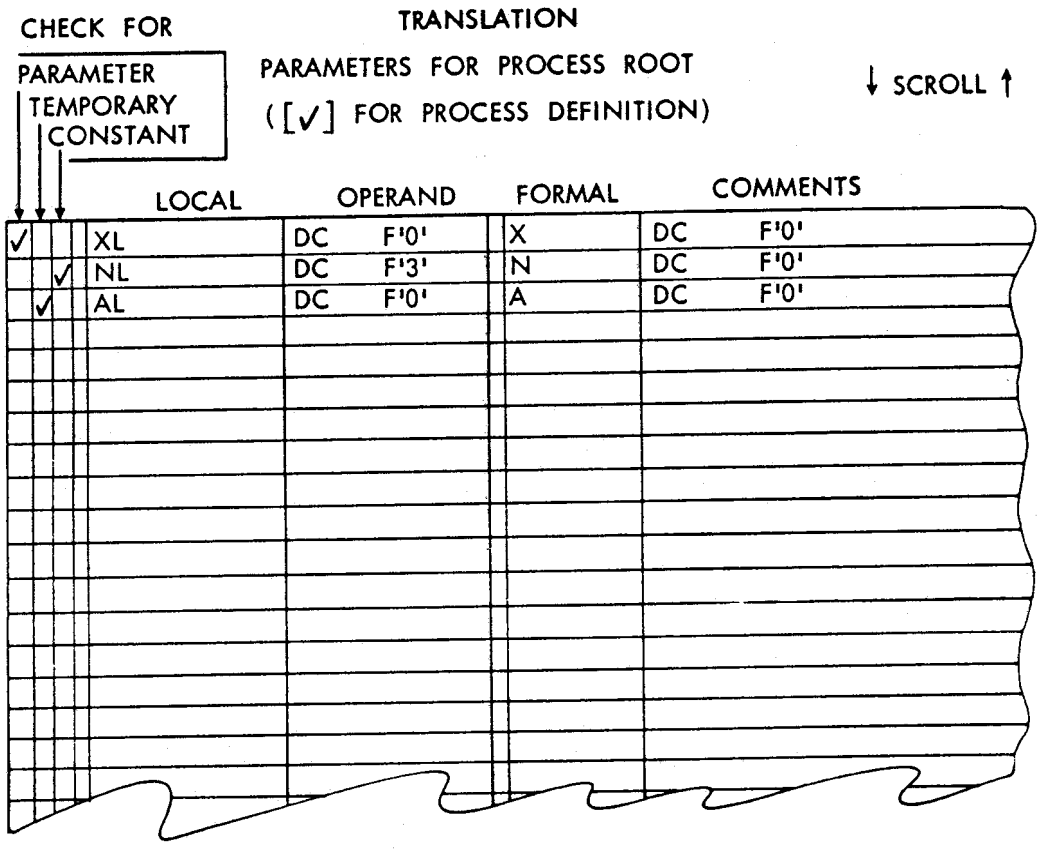


Fig. 5--Translation

parameter plane for data declarations, and associated with each instance of a closed process is a translation plane for specifying the data interface to the definition.

IV. DISCUSSION

This Memorandum highlights rather than details all aspects of GRAIL operations or of the flowchart language. The operating environment intended to support programs compiled from GRAIL flowcharts is described in Ref. 1.

The flowchart language describes a programming system and also serves as a graphical design problem. The capabilities of the language as a programming system were tested by writing GRAIL itself within the flowchart symbolism. The single exception was a supervisory process that interfaces with the hardware and supports some of the operations of GRAIL constructs.

Flowcharting presents interesting graphical manipulation and response problems aside from symbol recognition. Since the designers intended to have all input interpreted from stylus actions alone, some means of associating stylus position, flowchart symbols, and the man's intentions had to be determined. The primary technique employed surrounds the symbols, or other stylus-sensitive areas, with logical (invisible) rectangles whose sides are parallel to the display axes. These rectangles are used for rough sorting. Finer individual resolution in the context of the specific symbol determines whether a stroke *was inside of*, *on the boundary of*, or *had run into* a flowchart symbol. This technique becomes progressively less effective if the symbol is not rectangular or convex, or is rotated or linear. Although delaying a decision pending further stylus data is nearly always effective, the delay is used only for the *on the boundary of* case.

The system depends primarily on intensity changes, in addition to recognition and inking, for responses (except for the English-language message response). Other techniques were available (e.g., display data flashing), but tended to be distracting. A larger variety of distinguishable intensity levels, or perhaps colors, would avoid

context-dependent interpretation of these responses. However, the limited response classes and the area analyses proved quite effective for GRAIL purposes.

REFERENCES

1. Ellis, T. O., J. F. Heafner, and W. L. Sibley, *The GRAIL Project: An Experiment in Man-Machine Communications*, The RAND Corporation, RM-5999-ARPA, September 1969.

2. -----, *The GRAIL System Implementation*, The RAND Corporation, RM-6002-ARPA, September 1969.

3. Bernstein, M. I., and H. L. Howell, *Hand-Printed Input for On-Line Systems: Final Report for Phase I*, System Development Corporation, TM-(L)-3964/000/00, Santa Monica, California, April 1968.

4. Davis, M. R., and T. O. Ellis, *The RAND Tablet: A Man-Machine Graphical Communication Device*, The RAND Corporation, RM-4122-ARPA, August 1964. (Also, *Proceedings of the FJCC, 1964*, p. 325.)

5. Ellis, T. O., and W. L. Sibley, *On the Development of Equitable Graphic I/O*, The RAND Corporation, P-3415, July 1966.

6. -----, *On the Problem of Directness in Computer Graphics*, The RAND Corporation, P-3697, March 1968.

7. Groner, G. F., *Real-Time Recognition of Handprinted Text*, The RAND Corporation, RM-5016-ARPA, October 1966. (Also, *Proceedings of the FJCC, 1966*, p. 591.)

8. Licklider, J.C.R., and W. E. Clark, "On-Line Man-Computer Communications," *Proceedings of the SJCC, 1962*, p. 113.

9. Sutherland, E. I., "Sketchpad: A Man-Machine Graphical Communication System," *Proceedings of the SJCC, 1963*, p. 329.

10. Sutherland, W., *Semiannual Technical Summary Graphics*, Lincoln Laboratory, Lexington, Mass, November 30, 1967, pp. 24-27.

11. Sutherland, E. I., and R. F. Sproul, "A Clipping Divider," *Proceedings of the FJCC, 1968*, pp. 765-775.